Embedded Systems Projektarbeit

SoSe 2019

STM32F429 Discovery

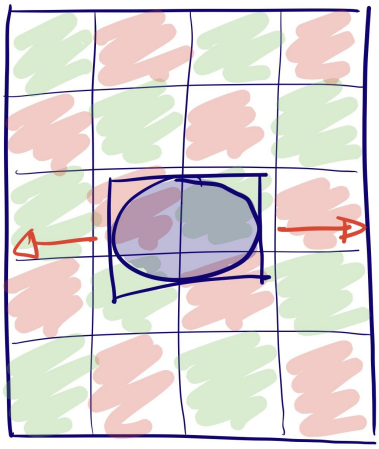
Anna Ivanov, Jens Kolz, Nadine Warneke

25.09.2019

1. Einleitung

In diesem Projekt arbeiten wir mit dem Microcontroller STM32F429 Discovery von STMicroelectronics. Dieser besitzt ein ILI9341 LCD-Display, auf dem wir zwei Layer erstellen.
Auf dem ersten Layer wird ein grün/rotes Schachbrett Muster erstellt und auf dem zweiten Layer ein hellblauer durchsichtiger Kreis, der auf dem Display kontinuierlich von links nach rechts wandert.
Für dieses Projekt dürfen nur in der Vorlesung bereitgestellte Dateien mit dem vorbereiteten Code und die IDE µVision 5 von Keil verwendet werden.

Zum Anfang wird der SPI initialisiert, um Befehle senden und um das Display ansteuern zu können.
Sobald dies erledigt ist, wird der LTDC initialisiert, der eine schnellere Verbindung gewährleistet und über diesen die Layer für das Display aufgebaut werden können.
Danach werden die Formen für die verschiedenen Layer vom DMA2D gezeichnet und über den SDRam geladen. Die Bewegung des Kreises wird ermöglicht durch das Verschieben des oberen Layers.



2. Vorgehensweise

2.1. SPI

Für die Kommunikation innerhalb des STM wird eine SPI Schnittstelle benötigt, die die Befehle versendet und so eine Verbindung zum Display aufbaut.
Der STM besitzt sechs SPI Schnittstellen mit ihren jeweiligen PINs.
Für das Ansteuern benötigt man den SPI5, da nur dieser eine Verbindung zu dem Display besitzt.
Der SPI5 verwendet die Pins 9 (MOSI) , 8 (MISO) und 7 (SCK) in Port F.
Auch wird der CSX und DCX benötigt:

- CSX Pin 2 Port C
- DXC Pin 13 Port D

Den CSX und DCX initialisieren wir als erstes, indem man zuerst den AHBClock von Port C und D aktiviert und mithilfe der folgenden Funktionen die Werte setzt:

```
//Init DCX
AHBClockEnable(PERIPHERY_AHB_GPIOD);
WrGpioMD(rbGPIOD, ILI9341_WRX, MASK_GPIO_MODER_GPOUT);
WrGpioSP(rbGPIOD, ILI9341_WRX, MASK_GPIO_OSPEEDR_MED);

//Init CS
AHBClockEnable(PERIPHERY_AHB_GPIOC);
WrGpioMD(rbGPIOC, ILI9341_CS, MASK_GPIO_MODER_GPOUT);
WrGpioSP(rbGPIOC, ILI9341_CS, MASK_GPIO_OSPEEDR_MED);

WrGpioSet(rbGPIOC, ILI9341_CS);
```

Danach kann man den SPI initialisieren.
Zuerst müssen die Clocks für SPI5 und Port F aktiviert werden und daraufhin über eine alternative Funktion die SPI Pins.

```
APBClockEnable(PERIPHERY_APB_SPI5);
AHBClockEnable(PERIPHERY_AHB_GPIOF);

static constexpr BYTE const pins[] = {7,8,9};
EnableAF(rbGPIOF, MASK_GPIO_AFR_AF05, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, pins, 3U);
```

Im Control Register des SPI5 müssen dann folgende Werte festgelegt werden:

```
rbSPI5.CR1 &= ~MASK_SPI_CR1_SPE;
rbSPI5.CR1 |= MASK_SPI_CR1_SSM;
rbSPI5.CR1 |= MASK_SPI_CR1_SSI;
rbSPI5.CR1 |= MASK_SPI_CR1_MSTR;
rbSPI5.CR1 |= MASK_SPI_CR1_SPE;
```

Wenn der SPI initialisiert wurde, kann man über den DRbyte Daten versenden und somit Befehle weiterleiten.

2.2. LTDC

Um den LTDC zu initialisieren benötigt man mehrere Schritte:

- Setzen der LTDC Pins

```
//GPIOA                                     Blue5, VSYNC, Green2, Red4, Red5
static constexpr BYTE const aPins[] =       { 3,      4,    6,    11,   12};
//GPIOB                                     Red3,Red6
static constexpr BYTE const bPinsFirst[] =  { 0,  1};
//GPIOB                                   Blue6, Blue7, Green4, Green5
static constexpr BYTE const bPinsSecond[] = {8,   9,      10,     11};
//GPIOC                                   HSYNC,Green6,Red2
static constexpr BYTE const cPins[] =       { 6,    7,  10};
//GPIOD                                   Green7,Blue2
static constexpr BYTE const dPins[] =       { 3,  6};
//GPIOF                                     Enable
static constexpr BYTE const fPins[] =       { 10};
//GPIOG                                   Red7, CLK, Blue3
static constexpr BYTE const gPinsFirst[] =  { 6,  7, 11};
//GPIOG                                     Green3, Blue4
static constexpr BYTE const gPinsSecond[] = { 10,   12};

APBClockEnable(PERIPHERY_APB_LTDC);
AHBClockEnable(PERIPHERY_AHB_GPIOA);
AHBClockEnable(PERIPHERY_AHB_GPIOB);
AHBClockEnable(PERIPHERY_AHB_GPIOC);
AHBClockEnable(PERIPHERY_AHB_GPIOD);
AHBClockEnable(PERIPHERY_AHB_GPIOF);
AHBClockEnable(PERIPHERY_AHB_GPIOG);

EnableAF(rbGPIOA, MASK_GPIO_AFR_AF14, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, aPins);
EnableAF(rbGPIOB, MASK_GPIO_AFR_AF09, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, bPinsFirst);
EnableAF(rbGPIOB, MASK_GPIO_AFR_AF14, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, bPinsSecond);
EnableAF(rbGPIOC, MASK_GPIO_AFR_AF14, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, cPins);
EnableAF(rbGPIOD, MASK_GPIO_AFR_AF14, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, dPins);
EnableAF(rbGPIOF, MASK_GPIO_AFR_AF14, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, fPins);
EnableAF(rbGPIOG, MASK_GPIO_AFR_AF14, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, gPinsFirst);
EnableAF(rbGPIOG, MASK_GPIO_AFR_AF14, MASK_GPIO_OSPEEDR_HIGH, MASK_GPIO_OTYPER_PP, gPinsSecond);
```

- Aktivieren der LTDC Clock im RCC Register
- Einstellen der Pixel Clock
- Berechnen der vertikalen und horizontalen Synchronisation

```
// VSync HSync setzen
rbLTDC.SSCR = ((hSync - 1) << INDX_LTDC_SSCR_HSW) | ((vSync - 1) << INDX_LTDC_SSCR_VSH);

// BackPorch
rbLTDC.BPCR = ((hSync + horBackPorch - 1) << INDX_LTDC_BPCR_AHBP) | ((vSync + verBackPorch - 1) << INDX_LTDC_BPCR_AVBP);

// Active Width / Height
rbLTDC.AWCR = ((hSync + horBackPorch + width - 1) << INDX_LTDC_AWCR_AAW) | ((vSync + verBackPorch + height - 1) << INDX_LTDC_AWCR_AAH);

// Total Width / Height
rbLTDC.TWCR = ((verTotal - 1) << INDX_LTDC_TWCR_TOTALH) | ((horTotal - 1) << INDX_LTDC_TWCR_TOTALW);
```

- Setzen der Polarität

Wenn der LTDC soweit initialisiert ist, kann der erste Layer eingerichtet werden.
Für das Schachbrettmuster muss man die Größe des Layers über das ganze Display festlegen:

```
//Layer 0
rbLTDC.Layer0.CACR = 255;
// Start und Stop pos für Weite Layer0
rbLTDC.Layer0.WHPCR = hSync + horBackPorch << INDX_LTDC_Lx_WHPCR_WHSTPOS | hSync + horBackPorch + width << INDX_LTDC_Lx_WHPCR_WHSPPOS;
// Start und Stop pos für Höhe Layer0
rbLTDC.Layer0.WVPCR = vSync + verBackPorch << INDX_LTDC_Lx_WVPCR_WVSTPOS | vSync + verBackPorch + height << INDX_LTDC_Lx_WVPCR_WVSPPOS;
```

Bei dieser Größe des Layers benötigt man eine 16 bit Farbe, da sonst die Bandbreite des STM komplett ausgelastet ist.
Für das Schachbrettmuster wird RGB565 verwendet:

```
rbLTDC.Layer0.PFCR |= MASK_LTDC_Lx_PFCR_RGB565;
```

Danach legt man fest in welchem Bereich der Layer im SDRam beginnt und wie viel Speichergröße er benötigt:

```
rbLTDC.Layer0.CFBAR = 0xD0000000U;
rbLTDC.Layer0.CFBLR |= ((width * 4) << 16 ) | (width * 4);
rbLTDC.Layer0.CFBLNR |= height;
```

Das Gleiche wird nun für den zweiten Layer getan.
Da dieser für den Kreis nicht das ganze Display benötigt, wird er kleiner gestaltet und mittig vom Display gesetzt:

```
rbLTDC.Layer1.CACR = 255;
// Start und Stop pos für Weite Layer1
rbLTDC.Layer1.WHPCR = hSync + horBackPorch + 70 << INDX_LTDC_Lx_WHPCR_WHSTPOS | hSync + horBackPorch + 170 << INDX_LTDC_Lx_WHPCR_WHSPPOS;
// Start und Stop pos für Höhe Layer1
rbLTDC.Layer1.WVPCR = vSync + verBackPorch + 110 << INDX_LTDC_Lx_WVPCR_WVSTPOS | vSync + verBackPorch + 210 << INDX_LTDC_Lx_WVPCR_WVSPPOS;
```

Für diesen Layer kann man nun ARGB8888 verwenden:

```
rbLTDC.Layer1.PFCR |= MASK_LTDC_Lx_PFCR_ARGB8888;
```

Der Bereich im SDRam muss nach dem ersten Layer beginnen, da sonst Fehler entstehen können. Dafür erhöht man die Startadresse um 320*240*4=307200:

```
rbLTDC.Layer1.CFBAR = 0xD0307200U;
rbLTDC.Layer1.CFBLR |= ((101* 4) << 16 ) | (101 * 4);
rbLTDC.Layer1.CFBLNR |= 101;
```

Wenn der zweite Layer erstellt wurde, machen wir diesen durchsichtig, indem im SDRam keine Farbe übergeben und der Alphawert auf 0 gesetzt wird:

```
int* addr = (int*)0xD0307200U;
for(int i = 0; i < (101 * 101 * 4);i++){
  *addr = 0x00000000;
  addr = addr + 1;
}
```

Nach dem Befüllen der Layer müssen diese und der LTDC aktiviert werden:

```
  //Reload Layer
rbLTDC.Layer0.CR |= MASK_LTDC_Lx_CR_LEN;
rbLTDC.Layer1.CR |= MASK_LTDC_Lx_CR_LEN;
rbLTDC.SRCR |= MASK_LTDC_SRCR_IMR;
//enable LTDC
rbLTDC.GCR |= MASK_LTDC_GCR_LTDCEN;
```

2.3. DMA2D

Um den DMA2D nutzen zu können, müssen die Clocks von DMA_2D, DMA1 und DMA2 aktiviert werden:

```
AHBClockEnable(PERIPHERY_AHB_DMA_2D);
AHBClockEnable(PERIPHERY_AHB_DMA1);
AHBClockEnable(PERIPHERY_AHB_DMA2);
```

Danach können über diesen die Formen generiert werden. In der drawRectangle Funktion wird die x und y Koordinate als Startpunkt des Rechtecks , wie hoch und breit das Rechteck sein soll, die Farbe, die Speicheradresse und Layerbreite des Layers übergeben. Danach wird diese Zeichnung in den SDRam geladen:

```
void Dma2d::drawRectangle(uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint32_t color, uint32_t memory, uint32_t displayWidth){
  rbDMA_2D.CR |= MASK_DMA2D_CR_MODE_REG2MEM;
  rbDMA_2D.OCOLR = color;

  Dma2d::SetMemory(memory + ((x + y * displayWidth)*4),displayWidth - width, width, height);

  rbDMA_2D.CR |= MASK_DMA2D_CR_START;

  Dma2d::waitToInit();
}
```

Innerhalb der Funktion wird der Speicherbereich für diese Zeichnung über die SetMemory Funktion festgelegt:

```
void Dma2d::SetMemory(uint32_t memoryAddr, uint32_t offset, uint32_t width, uint16_t height){
  rbDMA_2D.OMAR = memoryAddr;
  rbDMA_2D.OOR = offset;
  rbDMA_2D.NLR = ((width) << 16) | (height);
}
```

Die drawCircle Funktion startet wie auch beim Rechteck bei einer x und y Koordinate, von diesem Punkt erstellt die Funktion im Abstand des Radius einen Kreis mittels der drawRectangle Funktion:

```
void Dma2d::drawCircle(uint32_t x, uint32_t y, uint32_t radius, uint32_t color, uint32_t memory, uint32_t displayWidth)
{
  for(uint32_t i = 0; i < (radius * 2); i++)
    for(uint32_t j = 0; j< (radius * 2); j++)
    {
      if(((i - radius) * (i - radius) + (j - radius) * (j - radius)) <= radius * radius)
      {
        drawRectangle(j+x, i+y, 1, 1, color, memory, displayWidth);
      }
  }
}
```

Um das Zeichnen der Formen ohne Probleme zu gewährleisten, wird mithilfe der waitToInit Funktion abgewartet bis der Schreibvorgang beendet ist bevor ein neuer beginnen kann:

```
# define DMA2D_WORKING ((rbDMA_2D.CR & MASK_DMA2D_CR_START))
# define DMA2D_WAIT do{while(DMA2D_WORKING); rbDMA_2D.IFCR = MASK_DMA2D_IFCR_CTCIF;}while(0);

void Dma2d::waitToInit(){
   DMA2D_WAIT;

   rbRCC.AHB1RSTR |= MASK_RCC_AHB1_DMA1;
   rbRCC.AHB1RSTR &= ~MASK_RCC_AHB1_DMA1;

   rbDMA_2D.CR |= (WORD)INDX_DMA2D_CR_START;

   DMA2D_WAIT;
}
```

3. Umsetzung

Das Schachbrettmuster wird mit zwei for-Schleifen erzeugt, um sechs Rechtecke in der Breite und acht in der Höhe zu erzeugen. Die innere for-Schleife erzeugt für jede Zeile, Quadrate mit einer festgelegten Größe, in diesem Fall sechs Stück. Anschließend werden durch die äußere for-Schleife die Anzahl an Zeilen erzeugt, hier acht. Mithilfe einer weiteren Variable (z) wird mit jedem Schleifendurchlauf die Farbe des Quadrates geändert:

```
int x=0;
int y=0;
int z=1;
for(int i = 0; i < 8; i++){
  x=0;
  for(int j = 0; j < 6; j++){
    if(z % 2 == 0){
      dma.drawRectangle(x,y,20,40,RED,LAYER1,LAYER1_WIDTH);
    }else{
      dma.drawRectangle(x,y,20,40,GREEN,LAYER1,LAYER1_WIDTH);
    }
    z++;
    x+=20;
  }
  y+=40;
  z++;
}
```

Die Bewegung des Kreises wird mit einer boolean while-Schleife erzeugt, indem wir den Layer, der den Kreis beinhaltet im Ganzen verschieben. Sobald er den Bildschirmrand erreicht, wird die boolean Variable umgestellt und der Layer bewegt sich in die andere Richtung:

```
  x = 70;
  int x2 = 170;
  bool rechts = false;
  while(true){
    for(int delay = 1200000; delay > 0; delay--);
    rbLTDC.SRCR |= MASK_LTDC_SRCR_IMR;
    if(x2 < width && rechts){
    x++;
    x2++;
    rbLTDC.Layer1.WHPCR = hSync + horBackPorch + x << INDX_LTDC_Lx_WHPCR_WHSTPOS | hSync + horBackPorch +  x2 << INDX_LTDC_Lx_WHPCR_WHSPPOS;
    }else if(x > -1 && !rechts){
      x--;
      x2--;
      rbLTDC.Layer1.WHPCR = hSync + horBackPorch + x << INDX_LTDC_Lx_WHPCR_WHSTPOS | hSync + horBackPorch +  x2 << INDX_LTDC_Lx_WHPCR_WHSPPOS;
    }else  {
      rechts = !rechts;
    }
  }
}
```

Das Resultat sieht am Ende so aus: